Collision-Free Reactive Mission and Motion Planning for Multi-robot Systems

Jonathan A. DeCastro, Javier Alonso-Mora, Vasumathi Raman, Daniela Rus and Hadas Kress-Gazit

1 Introduction

We aim to synthesize correct-by-construction controllers for a team of robots performing high-level tasks that capture locomotion and actuation. Towards the goal of capable human-robot teams, the tasks we consider are reactive, requiring each robot to react and adapt to changes in the environment (e.g. the motion of other robots or people) at runtime. It has been demonstrated that reactive task specifications written in linear temporal logic (LTL) can be automatically converted into high-level plans that compose basic (atomic) actions to fulfill the task [10]. For example, consider two robots tasked with patrolling the rooms of a house in order to remove garbage and pick up misplaced toys. For high-level synthesis, the atomic actions "remove garbage" and "pick up toys" are assumed to be perfectly executable: they are treated as black boxes in a *discrete abstraction* implicit to the specified task. When a controller is synthesized for this high-level mission specification, it therefore does not govern the design of these low-level actions, nor the behavior of the

Jonathan DeCastro and Javier Alonso-Mora contributed equally to this work

J.A. DeCastro (✉) · H. Kress-Gazit
Cornell University, Ithaca, NY, USA
e-mail: jad455@cornell.edu

H. Kress-Gazit
e-mail: hadaskg@cornell.edu

J. Alonso-Mora (✉) · D. Rus
Massachusetts Institute of Technology, Cambridge, MA, USA
e-mail: jalonsom@mit.edu

D. Rus
e-mail: rus@mit.edu

V. Raman
California Institute of Technology, Pasadena, CA, USA
e-mail: vasu@caltech.edu

A. Bicchi and W. Burgard (eds.), *Robotics Research*, Springer Proceedings in Advanced Robotics 2, DOI 10.1007/978-3-319-51532-8_28

dynamic obstacles – e.g. the inhabitants of the household – that may interfere with their execution. In this work, we address the challenge of ensuring collision-freeness of the generated motion plans, bridging this disconnect between the high-level plan and the low-level actions.

Our approach efficiently abstracts collisions between the robots and dynamic obstacles, and automatically synthesizes a controller for each robot such that the team satisfies the high-level specification. Synthesis is made tractable using a local navigation controller for collision-avoidance, eliminating the need to explicitly model dynamic obstacles in the discrete abstraction. We show that we are able to preserve the global guarantees on task satisfaction using a local method for collision avoidance. This is significant because local planning methods are myopic, and usually do not yield global guarantees in multi-agent settings due to the threat of deadlock (a robot is unable to make forward progress) or livelock (the robot is trapped in an infinite cycle without ever achieving its goals).

Our method applies to the general case of motion planning tasks for multi-robot systems involving unmodeled and uncontrolled dynamic obstacles. We reduce the worst-case conservatism with respect to uncontrolled agents and dynamic obstacles that is typical of most approaches based on reactive synthesis (e.g. [15]). Our results have major implications on the scalability of controller synthesis for dynamic and partially-unmodeled environments.

1.1 Related Work

*High-level Reactive Synthesis.*Reactive synthesis offers a user-friendly approach to the control of complex robotic systems [10], and is especially compelling given the complex nature of multi-agent scenarios. Correct-by-construction reactive controllers have been extended with notions of optimality [15] and distributed teaming [5]. In most approaches, moving obstacles are modeled in a discrete manner as part of the abstraction, leading to over-conservative restrictions like requiring robots to be at least one region apart. Synthesis in dynamic environments thus presents a crucial dilemma: explicitly modeling the state of all other agents is computationally prohibitive, but incomplete models can obliterate the guarantees afforded by the planner. To address the state-explosion problem when modeling uncontrollable agents, [18] formulate an incremental procedure that adapts the number of agents considered in the plan depending on available computational resources. On the other hand, the authors in [12] make local modifications to the synthesized strategy when new elements of the environment are discovered that violate the original assumptions. In contrast to previous work on synthesis for multi-robot tasks, our method preserves guarantees via local collision avoidance, only requiring local awareness of the robot's surroundings. While we also update our specification in a systematic fashion, we do so offline (prior to execution), such that the synthesized strategies preserve guarantees at runtime. Our approach to providing feedback on failed specifications, described in Sect. 4.2, is inspired by recent formal approaches to automated assumption generation [3, 6, 11] and explaining the cause of failure [13].

Collision Avoidance. Online reactive methods, such as [2], typically do not provide global mission fulfillment guarantees. We leverage and extend this work to enforce collision avoidance and motion constraints in the short time horizon, while relying on the high-level planner for guidance to fulfill the global mission.

1.2 Contribution

Our contribution is a holistic synthesis approach that leverages high-level mission planning and low-level motion planning to provably achieve collision-free high-level behaviors in dynamic environments. Local planning capabilities are abstracted in a manner that allows dynamic obstacles to remain unmodeled at the high level during synthesis, and the high level provides deadlock resolution strategies that ensure task satisfaction.

We further contribute:

(a) Automatic feedback-generation for revising specifications. We automatically generate human-comprehensible assumptions in LTL that, if satisfied by the controlled robots and the dynamic obstacles, would ensure correct behavior.
(b) An optimization-based method that extends [2] for synthesizing controllers that guarantee real-time collision avoidance with static and dynamic obstacles in 3D environments while remaining faithful to the robot's dynamics.

Experimental results with ground robots and simulated quadrotors are discussed.

2 Preliminaries

Scalars are denoted by x and vectors by $\mathbf{x} \in \mathbb{R}^n$, with n denoting the dimension of the workspace. The robot's current position is denoted by $\mathbf{p} \in \mathbb{R}^n$ and its current velocity by $\mathbf{v} = \dot{\mathbf{p}}$. A map of the workspace $W \subset \mathbb{R}^n$ is considered, and formed by a set of static obstacles (given by a list of polytopes) $\mathcal{O} \subset \mathbb{R}^n$. For high-level synthesis, the map is abstracted by a set of discrete regions $\mathcal{R} = \{R_1 \dots R_p\}$ covering the map W, where the open sets $R_\alpha \subseteq W$.

2.1 Linear Temporal Logic

LTL formulas are defined over the set AP of atomic (Boolean) propositions by the recursive grammar $\varphi ::= \pi \in AP \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \bigcirc\varphi \mid \varphi_1 \mathcal{U} \varphi_2$. From the Boolean operators $\wedge$ "conjunction" and $\neg$ "negation", and the temporal operators $\bigcirc$ "next" and $\mathcal{U}$ "until", the following operators are derived: "disjunction" $\vee$, "implication" $\Rightarrow$, "equivalence" $\Leftrightarrow$, "always" $\Box$, and "eventually" $\Diamond$. Refer to [16] for a description of the semantics of LTL. Let AP represent the set of atomic propositions, consisting of *environment* propositions ($\mathcal{X}$) corresponding to thresholded sensor values, and *system*

propositions ($\mathcal{Y}$) corresponding to the robot's actions and location with respect to a partitioning of the workspace. The value of each $\pi \in \mathcal{X} \cup \mathcal{Y}$ is the abstracted binary state of a low-level component.

Definition 1 (*Reactive Mission Specification*) A *Reactive Mission Specification* is an LTL formula of the form $\varphi = \varphi_i^e \wedge \varphi_t^e \wedge \varphi_g^e \implies \varphi_i^s \wedge \varphi_t^s \wedge \varphi_g^s$, with s and e standing for 'system' and 'environment', such that

- φ_i^e, φ_i^s are formulas for the *initial conditions* free of temporal operators.
- φ_t^e, φ_t^s are the *safety conditions* (transitions) to be satisfied always, and are of the form $\square\psi$, where ψ is a Boolean formula over $AP \cup \bigcirc AP$.
- φ_g^e, φ_g^s are the *liveness conditions* (goals) to be satisfied infinitely often, with each taking the form $\square\diamondsuit\psi$, with ψ a Boolean formula over $AP \cup \bigcirc AP$.

A strategy automaton that *realizes* a reactive mission specification φ is a deterministic strategy that, given a finite sequence of truth assignments to the variables in $\mathcal{X}$ and $\mathcal{Y}$, and the next truth assignment to variables in $\mathcal{X}$, provides a truth assignment to variables in $\mathcal{Y}$ such that the resulting infinite sequence satisfies φ. If such a strategy can be found, φ is *realizable*. Otherwise, it is *unrealizable*. Strategy automata for φ of the form above can be synthesized [4], and converted into hybrid controllers for robotic systems by invoking atomic controllers [10]. These controllers are *reactive*: they respond to sensor events at runtime.

2.2 LTL Encoding for Multi-robot Tasks

We adopt an LTL encoding that is robust to the inherent variability in the duration of inter-region robot motion in continuous environments [14]. Let $AP_{\mathcal{R}} = \{\pi_\alpha^i \mid R_\alpha \in \mathcal{R}\}$ be the set of Boolean propositions representing the workspace regions, such that $\pi_\alpha^i \in AP_{\mathcal{R}}$ is `True`when robot i is physically in R_α for $\alpha \in [1, p]$. We call π_α^i in $AP_{\mathcal{R}} \subseteq \mathcal{X}$ a *completion* proposition, signaling when robot i reaches R_α. We also define the set $AP_{\mathcal{R}}^{act} \subseteq \mathcal{Y}$ that captures robot commands that *initiate* movement between regions. We call $\pi_{act,\alpha}^i$ in $AP_{\mathcal{R}}^{act}$ an *activation* variable for moving to R_α. Non-motion actions are handled similarly.

Definition 2 (*LTL Encoding of Motion* [14]) A task encoding that admits arbitrary controller execution durations is

$$\psi_t^s: \bigwedge_{\substack{\pi_\alpha^i \in AP_{\mathcal{R}}, \\ i \in [1, n_{robots}]}} \square(\bigcirc\pi_\alpha^i \Rightarrow \bigvee_{R_\beta \in Adj(R_\alpha)} \bigcirc\pi_{act,\beta}^i),$$

$$\psi_t^e: \bigwedge_{\substack{\pi_\alpha^i \in AP_{\mathcal{R}}, \\ R_\beta \in Adj(R_\alpha), \\ i \in [1, n_{robots}]}} \square(\pi_\alpha^i \wedge \pi_{act,\beta}^i \Rightarrow \bigcirc\pi_\alpha^i \vee \bigcirc\pi_\beta^i),$$

$$\psi_g^e: \square\diamondsuit \bigwedge_{\substack{i \in [1, n_{robots}] \\ \pi_{act,\alpha}^i \in AP_{\mathcal{R}}^{act} \cup AP_{\mathcal{A}}^{act}}} \left(\left(\pi_{act,\alpha}^i \wedge \bigcirc(\pi_\alpha^i \vee \neg\pi_{act,\alpha}^i)\right) \vee \left(\neg\pi_{act,\alpha}^i \wedge \bigcirc(\neg\pi_\alpha^i \vee \pi_{act,\alpha}^i)\right)\right),$$

where $Adj : \mathcal{R} \rightarrow 2^{\mathcal{R}}$ is an adjacency relation on regions in $\mathcal{R}$ and n_{robots} is the number of robots. The ψ_t^s-formula is a system safety condition describing which actions can occur ($\bigcirc\pi^i_{act,\beta}$) given the observed completion variables ($\bigcirc\pi^i_{\alpha}$). Formula ψ_t^e captures the allowed transitions ($\bigcirc\pi^i_{\beta}$) given past completion (π^i_{α}) and activation ($\pi^i_{act,\beta}$) variables. Formula ψ_g^e enforces that every motion and every action eventually completes as long as the activation variable is held fixed. Both ψ_t^e and ψ_g^e are included as conjuncts to the antecedent of φ.

2.3 Local Motion Planning and Robot Dynamics

A collision-free local motion for each robot is computed independently and online based on the current transition in the strategy automaton. We build on the work on distributed Reciprocal Velocity Obstacles with motion constraints [1], and its recent extension to aerial vehicles [2]. Letting $t \in \mathbb{R}_+$ denote time and t_k the current time instant, we define the relative time $\tilde{t} = t - t_k \in [0, \infty)$ and the time horizon of the local planner $\tau > 0$, greater than the required time to stop.

For a robot, a set of candidate local trajectories is considered, each defined by $\mathbf{p}^{robot}(\tilde{t}) = f(\mathbf{z}, \mathbf{u}, \tilde{t})$, continuous in the initial state $\mathbf{z} = [\mathbf{p}, \dot{\mathbf{p}}, \ddot{\mathbf{p}}, \dots]$ of the robot, respecting its dynamic constraints and given by an appropriate controller converging to a straight-line reference trajectory $\mathbf{p}_{\text{ref}}(\tilde{t}) = \mathbf{p} + \mathbf{u}\tilde{t}$ of constant velocity $\mathbf{u} \in \mathbb{R}^n$ and starting at the current position $\mathbf{p}$ of the robot. Local trajectories are now parametrized by $\mathbf{u}$. Suitable controllers include LQR and second order exponential curves, for ground robots [1] and quadrotors [2]. For a given robotic platform and controller, initial state $\mathbf{z}$ and reference velocity $\mathbf{u}$, the maximum deviation (initial position independent) between the reference and the simulated trajectory is

$$\gamma(\mathbf{z}, \mathbf{u}) = \max_{\tilde{t}>0} ||(\mathbf{p} + \tilde{t}\mathbf{u}) - f(\mathbf{z}, \mathbf{u}, \tilde{t})||_2. \tag{1}$$

Maximal errors $\gamma(\mathbf{z}, \mathbf{u})$ are precomputed, and stored for on-line use, for the low-level controller $f(\mathbf{z}, \mathbf{u}_i, \tilde{t})$ and a discretization of initial states $\mathbf{z}$ and reference velocities $\mathbf{u}$.

The idea of the method is as follows: (a) the radius of the robot is enlarged by a value $\varepsilon > 0$ for collision avoidance, computed with respect to the reference trajectories $\mathbf{p} + \mathbf{u}t$ and (b) the local trajectories are limited to those with a tracking error below ε with respect to their reference trajectory. At each time-step an optimal reference velocity $\mathbf{u}^* \in \mathbb{R}^n$ is obtained by solving a convex optimization in $\mathbb{R}^n$. The associated local trajectory is collision-free, satisfies the motion constraints and minimizes the deviation to a preferred velocity $\bar{\mathbf{u}}$.

We approximate robots by their smallest enclosing cylinder of radius r and height $2h$, denoted by V, and its ε-additive dilation of radius $\bar{r} = r + \varepsilon$ and height $\bar{h} = h + \varepsilon$ by V_ε, and assume that all dynamic obstacles maintain a constant velocity during the planning horizon, or cooperate in avoiding collisions.

3 Problem Formulation and Approach

Example 1 Consider the workspace in Fig. 1a, where two robots are tasked with visiting regions G1 and G2 infinitely often, $\varphi_s^g = \bigwedge_{i\in\{1,2\}} \square\diamondsuit(\pi_{G1}^i) \wedge \square\diamondsuit(\pi_{G2}^i)$.

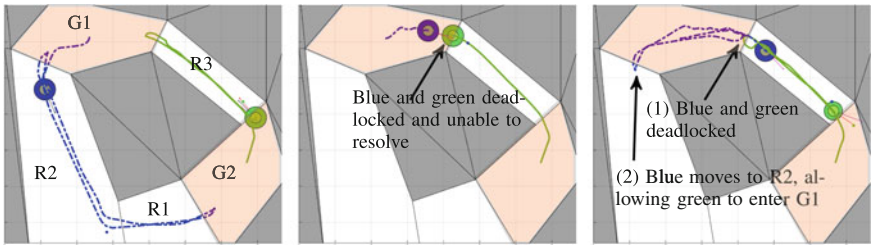
Figure 1 shows three approaches for solving this task. A simplistic approach is given in (a), where the robots do not have any collision avoidance and must always be one region apart from one another. The result is thus conservative; in fact, if any one region is blocked, the spec would be unrealizable. As will be shown in Sect. 7 this approach does not scale in the presence of dynamic obstacles. In (b), the robots employ a local planner to avoid collisions, along with a high-level controller that is less conservative but ignores deadlock. In this case, the execution fails to satisfy the task when the two robots become deadlocked. (c) shows our approach, where both robots are able to resolve encountered deadlocks under the synthesized integrated controller. The strategy can exploit the use of other regions if deadlock occurs.

3.1 Problem Formulation

Problem 1 (*Local Collision Avoidance*) For each robot of the team, construct an online local planner that respects the dynamics of the robot and guarantees collision avoidance with static and dynamic (moving) obstacles.

Problem 2 (*Synthesis of High-level Controller with Deadlock Resolution*) Given a topological map, a local motion planner that solves Problem 1 and a realizable mission specification φ that ignores collisions, automatically construct a specification φ' that models deadlock between robots and unmodeled dynamic obstacles and synthesize a controller that satisfies φ'.

By solving Problem 2, we guarantee avoiding deadlocks, but possibly at the sacrifice of task fulfillment. We therefore synthesize *environment assumption revisions*



(a) Mutually exclusive (b) Local planner (c) Integrated approach

Fig. 1 Three examples of motion planning, where the *blue* and *green* robots are initially in G1 and G2, respectively. **a** is the case with a global planner with no local planner; **b** and **c** correspond to the specifications φ (no deadlock resolution) and φ'' (with deadlock resolution) respectively

(additional LTL formulas) to identify detrimental cases where dynamic obstacles may trigger deadlock and trap the system from achieving its goals. These formulas are significant because they offer conditions upon which the environment must adhere to in order for the robot team to guarantee the task. As such, they must be clearly explained to the user. An example of such a condition is: "the environment will never cause deadlock if robot 1 is in the kitchen and moving to the door".

Problem 3 (***Revising Environment Assumptions***) Given an *unrealizable* reactive mission specification φ', synthesize environment assumption revisions $[\varphi_t^e]^{rev}$ such that the specification φ'' formed by replacing φ_t^e with $[\varphi_t^e]^{rev}$ is realizable, and provide the user with a human-readable description of these revisions.

3.2 Approach

We present a two-part solution to Problems 1, 2 and 3. Figure 2 shows the offline and online components and their interconnections; we now describe them in detail.

Offline. Given a high-level specification that takes a discrete topological map of the workspace and ignores collisions, a centralized controller is synthesized that considers possible deadlocks, iteratively revising the environment assumptions as necessary until a such a controller is synthesized. We also adopt a recovery scheme [17] that synthesizes a strategy that allows violations of environment *safety* assumptions to be tolerated, retaining satisfaction guarantees as long as the violation is transient. The automaton is agnostic to the robot's dynamics, which are instead accounted for by the local planner. The offline high-level synthesis is described in Sect. 4.

Online. At each time step of the execution, the synthesized automaton provides a desired goal for each controlled robot. Each robot independently computes a local trajectory that achieves its goal while avoiding other agents. If a deadlock is sensed, an alternative goal is extracted for some robot; the existence of such an alternative in the automaton is guaranteed by construction. The online local planner builds on [2] by adopting a convex optimization approach as described in Sect. 5.

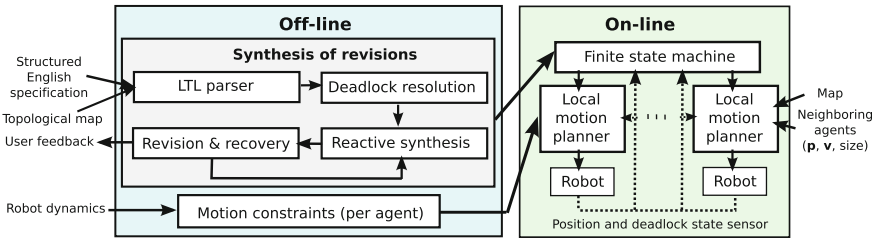


Fig. 2 Structure of the proposed mission planner, with offline and online parts

4 Offline Synthesis of the High-Level Mission Plan

We consider a high-level task specification, and a topological map of the environment composed of regions. This task specification φ ignores collisions and may result in deadlocks as in Fig. 1b. We modify the input specification with additional behaviors that the robot can take to resolve deadlock. The synthesized automaton guarantees completion of the task while redirecting the robots whenever deadlock occurs.

4.1 Deadlock Resolution

We define physical deadlock to be a situation where at least one robot has not reached its goal but cannot move. This can happen when an agent becomes blocked either by another agent or by a dynamic obstacle. To allow the high-level controller to resolve deadlock, we define a Boolean input signal $x^i \in \mathcal{X}$ that declares when a robot is in deadlock, where $i = 1, \ldots, n_{robots}$. We use the term *singleton deadlock* to refer to the specific case where a robot is in proximity of a dynamic obstacle. Additionally, define $x^{ij} \in \mathcal{X}$ to be an input signal that is `True` when two robots are in *pairwise deadlock* (both in deadlock and within a certain distance of one another), and `False` otherwise. We introduce the following shorthand:

$$\begin{aligned}
\theta_P^{ij} &= \neg x^{ij} \wedge \bigcirc x^{ij} && \text{rising edge–pairwise deadlock between robots } i \text{ and } j \\
\theta_S^{i} &= \neg x^{i} \wedge \bigcirc x^{i} && \text{rising edge–singleton deadlock for robot } i \\
\psi_{\alpha\beta}^{i} &= \pi_\alpha^i \wedge \bigcirc \pi_\alpha^i \wedge \pi_{act,\beta}^i && \text{incomplete transition } (\alpha \neq \beta)\text{; remain in region } (\alpha = \beta)
\end{aligned}$$

Resolving deadlock by redirecting the robot's motion based on the instantaneous value of x^i or x^{ij} alone may result in livelock, where the robot may be trapped away from its goals as a result of an alternating deadlock status. For this reason, our scheme automatically introduces additional memory propositions that are set when deadlock is sensed, and reset once the robot leaves its current region. While adding these propositions increases the state space of the synthesis problem, the advantage is that the robot can remember that deadlock had occurred and actively alter its strategy to overcome that situation. For each robot, we introduce the system propositions $\{y_\beta^i \mid R_\beta \in \mathcal{R}\} \subset \mathcal{Y}$ to represent the *memory* of deadlock occurring when activating a transition from a given region to region R_β.

$$\square \bigwedge_{\substack{\pi_\alpha^i \in AP_{\mathcal{R}}, \\ R_\beta \in Adj(R_\alpha)}} \left(y_\beta^i \wedge \pi_\alpha^i \implies \bigcirc(\neg \pi_{act,\alpha}^i \wedge \neg \pi_{act,\beta}^i) \right). \tag{2}$$

The role of y_β^i is to disallow the current transition (from R_α to R_β), as well as the self-transition from R_α to R_α. The self-transition is disallowed to force the robot

to leave the region where the deadlock occurred (R_α), instead of waiting for it to resolve; R_β is disallowed since the robot cannot make that transition.

Next, we enforce conditions to retain memory of *singleton deadlock*:

$$\bigwedge_{\substack{\pi_\alpha^i \in AP_{\mathcal{R}}, \\ R_\beta \in Adj(R_\alpha)}} \left(\neg y_\beta^i \Rightarrow \left((\theta_S^i \wedge \psi_{\alpha\beta}^i) \Rightarrow \bigcirc y_\beta^i\right)\right) \quad \text{and} \quad \bigwedge_{\substack{\pi_\alpha^i \in AP_{\mathcal{R}}, \\ R_\beta \in Adj(R_\alpha)}} \left(y_\beta^i \Rightarrow \left((\pi_\alpha^i \wedge \bigcirc \pi_\alpha^i) \Leftrightarrow \bigcirc y_\beta^i\right)\right). \tag{3}$$

The first formula sets the memory of deadlock y_β^i if the robot is activating transition from R_α to R_β. The second formula keeps memory set until a transition has been made out of R_α (to a region different from R_β).

For *pairwise deadlock*, we add the following conditions to set the memory proposition for at least one robot (at least one of the two robots reacts to the deadlock):

$$\square\left(\theta_P^{ij} \implies \left(\bigvee_{\ell \in \{i,j\}} \bigwedge_{\substack{\pi_\alpha^\ell \in AP_{\mathcal{R}}, \\ R_\beta \in Adj(R_\alpha)}} (\neg y_\beta^\ell \wedge \psi_{\alpha\beta}^\ell) \implies \bigcirc y_\beta^\ell\right)\right). \tag{4}$$

We also add the following to ensure that the memory propositions are only set when the rising edge of deadlock (singleton or pairwise) is sensed.

$$\square\left(\bigwedge_{\substack{i \in [1, n_{robots}] \\ R_\beta \in \mathcal{R}}} \left(\neg y_\beta^i \wedge \neg \theta_S^i \wedge \bigwedge_{\substack{j \in [1, n_{robots}] \\ j \neq i}} \neg \theta_P^{ij}\right) \implies \bigcirc \neg y_\beta^i\right). \tag{5}$$

In practice, we do not need a proposition y_β^i for every $\mathcal{R}_\beta \in \mathcal{R}$, but only $d = \max_{R_\alpha \in \mathcal{R}} (|Adj(R_\alpha)|)$ such propositions for each robot in order to remember all of the deadlocks around each region of the workspace. The number of conjuncts required for condition (4) is $\binom{n_{robots}}{2}$, but this has no effect on scalability since the runtime of the synthesis algorithm is only a function of the number of propositions and not the size of the specification.

Conjuncting the conditions (2)–(5) with φ_t^s yields a modified formula $[\varphi_t^s]'$ over the set AP, and the new specification $\varphi' = \varphi_i^e \wedge \varphi_t^e \wedge \varphi_g^e \implies [\varphi_i^s]' \wedge [\varphi_t^s]' \wedge \varphi_g^s$, where the initial conditions are modified by setting additional propositions x^i, y_α^i to false.

4.2 Specification Revisions

If the above specification φ' is synthesizable, Problem 2 is solved (for a proof, see Sect. 6). However, if the added restrictions to the system behavior result in the specification being unrealizable, Problem 3 must be solved by finding a set of assumptions on deadlock under which the environment must follow for the task to be guaranteed.

These assumptions are then presented to the user as a certificate of the conditions under which the guarantees for deadlock and livelock avoidance hold.

When a specification is unrealizable, there exist environment behaviors (called *environment counterstrategies*) that prevent the system from achieving its goals safely. Here we build upon the work of [3, 6, 11], processing synthesized counterstrategies to mine the necessary assumptions. Rather than synthesize assumptions from the counterstrategy, we instead search the counterstrategy for all deadlock occurrences, then store the corresponding conditions as assumptions.

We denote $\mathcal{C}_{\varphi'}$ as a state machine representing the counterstrategy for φ' and $\mathcal{Q}$ as the set of states for $\mathcal{C}_{\varphi'}$. To find the graph cuts in the counterstrategy graph that prevent the environment from impeding the system, we first define the following propositional representation of state $q \in \mathcal{Q}$ as $\psi(q) = \psi_{\mathcal{X}}(q) \wedge \psi_{\mathcal{Y}}(q)$, where

$$\psi_{\mathcal{Y}}(q) = \bigwedge_{\pi \in \gamma_{\mathcal{Y}}(q)} \pi \wedge \bigwedge_{\pi \in \mathcal{Y} \setminus \gamma_{\mathcal{Y}}(q)} \neg\pi, \quad \psi_{\mathcal{X}}(q) = \bigwedge_{\pi \in \gamma_{\mathcal{X}}(q)} \pi \wedge \bigwedge_{\pi \in \mathcal{X} \setminus \gamma_{\mathcal{X}}(q)} \neg\pi.$$

Next, the set of *cut transitions* S_{cuts} is computed as $S_{cuts} = \{(p, q) \in \mathcal{Q}^2 \mid q \in \delta(p), \psi(p)\psi(q) \models \bigvee_{i \in [1, n_{robots}]} \bigcirc \theta_S^i\}$, where $\delta : \mathcal{Q} \times 2^{\mathcal{Y}} \to 2^{\mathcal{Q}}$ is a transition relation returning the set of possible successor states given the current state and valuations of robot commands in $\mathcal{Y}$. S_{cuts} collects those transitions on which the environment has intervened (by setting deadlock) to prevent the system from reaching its goals.

Finally, the following safety assumptions are found:

$$\varphi_{rev}^e = \Box \bigwedge_{(p,q) \in S_{cuts}} (\psi_{\mathcal{Y}}(p) \wedge \psi_{\mathcal{X}}(p) \implies \neg \bigcirc \psi_{\mathcal{X}}(q)) \tag{6}$$

If any of the conjuncts in (6) falsify the environment, they are discarded. Then, set $[\varphi_t^e]^{rev} = \varphi_t^e \wedge \varphi_{rev}^e$ and construct the final specification $\varphi'' = \varphi_i^e \wedge [\varphi_t^e]^{rev} \wedge \varphi_g^e \implies [\varphi_i^s]' \wedge [\varphi_t^s]' \wedge \varphi_g^s$.

Algorithm 1 expresses our proposed approach for resolving deadlock. The automatically generated assumptions act to restrict the behavior of the dynamic obstacles. Each revision of the high-level specification excludes at least one environment move in a given state. Letting $|\cdot|$ denote set cardinality, with $2^{|\mathcal{X}|}$ environment actions and $2^{|\mathcal{Y}|}$ states, at most $2^{(|\mathcal{Y}|+|\mathcal{X}|)}$ iterations occur, though in our experience far fewer are needed. The generated assumptions are minimally restrictive – omitting even one allows the environment to cause deadlock, resulting in unrealizability.

Algorithm 1 Find realizable φ'' fulfilling task φ and resolving deadlock

$\varphi' \leftarrow \varphi_i^e \wedge \varphi_t^e \wedge \varphi_g^e \implies [\varphi_i^s]' \wedge [\varphi_t^s]' \wedge \varphi_g^s$
$[\varphi_t^e]^{rev} \leftarrow \varphi_t^e$; $\varphi'' \leftarrow \varphi_i^e \wedge [\varphi_t^e]^{rev} \wedge \varphi_g^e \Rightarrow [\varphi_i^s]' \wedge [\varphi_t^s]' \wedge \varphi_g^s$
while φ'' is *unrealizable* **do**
    Extract $\mathcal{C}_{\varphi''}$ from φ''
    $\varphi_{rev}^e \leftarrow$ Eq. (6)
    for each kth conjunct of φ_{rev}^e s.t. $\varphi_{rev}^e[k] \wedge [\varphi_t^e]^{rev} \neq$ `False` **do**
        Parse $\varphi_{rev}^e[k]$ into human-readable form and display to user.
        $[\varphi_t^e]^{rev} \leftarrow [\varphi_t^e]^{rev} \wedge \varphi_{rev}^e[k]$; $\varphi'' \leftarrow \varphi_i^e \wedge [\varphi_t^e]^{rev} \wedge \varphi_g^e \Rightarrow [\varphi_i^s]' \wedge [\varphi_t^s]' \wedge \varphi_g^s$
    end for
end while

5 Online Local Motion Planning

The result of the computation of Sect. 4 is a finite state machine where the states are labeled by regions and the transitions represent actions within the allowed navigation path. Each robot executes the finite state machine controller such that the overall multi-robot system is guaranteed to be livelock and collision free and deadlocks are resolved (may they appear). In this section we describe the local planner that links the high-level mission plan with the physical robot (recall Fig. 2). At each step of the online execution, the synthesized finite state machine provides a desired goal position for each robot and a preferred velocity $\bar{\mathbf{u}} \in \mathbb{R}^n$ towards it.

5.1 Constraints

To define the motion and inter-agent avoidance constraints we build on the approach in [2]. We additionally introduce constraints for avoiding static obstacles. For completeness, we give an overview of each of the constraints.

Motion constraints. Recalling Eq. (1) the motion constraint is given by the reference velocities for which the tracking error is below ε, $R(\mathbf{z}, \varepsilon) = \{\mathbf{u} \mid \gamma(\mathbf{z}, \mathbf{u}) \leq \varepsilon\}$, approximated by the largest inscribed convex polytope/ellipsoid $\hat{R}(\mathbf{z}, \varepsilon) \subset R(\mathbf{z}, \varepsilon)$.

Avoidance of other agents. Denote by $\mathbf{p}_j$, $\mathbf{v}_j$, $\bar{r}_j$ and $\bar{h}_j$ the position, velocity, dilated radius and height of a neighboring agent j. Assume that it keeps its velocity constant for $\tilde{t} \leq \tau$, for reciprocity see [2]. For every neighboring agent j, the constraint is given by the reference velocities $\mathbf{u}$ for which the agents' enveloping shape do not intersect within the time horizon. For cylindrically-shaped agents moving in 3D the velocity obstacle of colliding velocities is a truncated cone $VO_j^\tau =$

$$\{\mathbf{u} \mid \exists \tilde{t} \in [0, \tau] : \|\mathbf{p}^H - \mathbf{p}_j^H + (\mathbf{u}^H - \mathbf{v}_j^H)\tilde{t}\| \leq \bar{r} + \bar{r}_j,\ |p^V - p_j^V + (u^V - u_j^V)\tilde{t}| \leq \bar{h} + \bar{h}_j\},$$

where $\mathbf{p} = [\mathbf{p}^H, p^V]$, with $\mathbf{p}^H \in \mathbb{R}^2$ its projection onto the horizontal plane and $p^V \in \mathbb{R}$ its vertical component. The constraint is linearized to $A_j(\mathbf{p}, \varepsilon) = \{\mathbf{u} \mid \mathbf{n}_j^T \mathbf{u} \leq b_j\}$, where $\mathbf{n}_j \in \mathbb{R}^3$ and $b_j \in \mathbb{R}$ maximize $\mathbf{n}_j^T \mathbf{v} - b_j$ subject to $A_j(\mathbf{p}, \varepsilon) \cap VO_j^\tau = \emptyset$.

Avoidance of static obstacles. We extend a recent fast iterative method to compute the largest convex polytope in free space [7], by directing the growth of the region in the preferred direction of motion and enforcing that both the current position of the robot and a look ahead point in the preferred direction of motion are within the region. The convex polytope is computed in position space ($\mathbb{R}^3$ for aerial vehicles) and then converted to an equivalent region in reference velocity space. The details are given in Algorithm 2, where *directedEllipsoid*($\mathbf{p}$, $\mathbf{q}$) is the ellipsoid with one axis given by the segment $\mathbf{p} - \mathbf{q}$ and the remaining axis infinitesimally small.

Algorithm 2 Largest collision-free directed convex polytope

$L \leftarrow \mathbf{p} + \bar{\mathbf{u}}\{\tau, 0.7\tau, 0.5\tau, ..., 0\}$; $\mathbf{q} \leftarrow L[0]$; $L := L \setminus \mathbf{q}$; $P := \emptyset$
while $L \neq \emptyset$ and $\mathbf{p}, \mathbf{q} \notin P$ **do**
  $E \leftarrow directedEllipsoid(\mathbf{p}, \mathbf{q})$
  while not converged **do** // Largest polytope seeded in E computed as in [7]
    $P \leftarrow$ separating planes of E and dilated $\mathcal{O}$ (Quadratic program) , $P \subset \mathbb{R}^n \setminus (\mathcal{O} + V_\varepsilon)$
    If $\mathbf{p}, \mathbf{q} \notin P$ **then** { $\mathbf{q} \leftarrow L[0]$; $L := L \setminus \mathbf{q}$; **break**; }
    $E \leftarrow$ ellipsoid $E \subset P$ of maximal volume (Semi-Definite Program)
  end while
end while
$F(\mathbf{p}, \varepsilon) := (P - \mathbf{p})/\tau$ // Converts to ref. velocity, $\mathbf{u}$, space

5.2 Optimization

The optimization cost is given by two parts. The first part is a regularizing term penalizing changes in velocity (weighted by design constant $\bar{\alpha}$); the second minimizes the deviation to a preferred velocity, corrected by a repulsive velocity $\mathring{\mathbf{u}}$ inversely proportional to the distance to neighboring obstacles [2] when in close proximity. A convex optimization with quadratic cost and mixed linear/quadratic constraints is solved:

$$\mathbf{u}^* := \arg\min_{\mathbf{u}\in\mathbb{R}^n} (\bar{\alpha}||\mathbf{u} - \mathbf{v}||^2 + ||\mathbf{u} - (\bar{\mathbf{u}} + \mathring{\mathbf{u}})||^2), \tag{7}$$
$$\text{s.t.} \quad \mathbf{u} \in \hat{R}(\mathbf{z}, \varepsilon) \cap F(\mathbf{p}, \varepsilon) \quad \text{and} \quad \mathbf{n}_j^T \mathbf{u} \leq b_j \quad \forall j \text{ neighboring agent}$$

The solution of this optimization is a collision-free reference velocity $\mathbf{u}^*$ which minimizes the deviation towards the goal specified by the high-level state machine. The associated trajectory (see Sect. 2.3) is followed by the robot and is collision-free.

6 Guarantees

We provide proofs for the guarantees inherent to our synthesized controller.

Respects the modeled robot dynamics. By construction of the local planner, the controller is guaranteed correct with respect to the low-level controller $f(\mathbf{z}, \mathbf{u}, \tilde{t})$, which is continuous on the initial state of the robot and respects its dynamics.

Yields collision-free motion. If (7) is *feasible*, collision-free motion is guaranteed for the local trajectory up to time τ (the optimal reference trajectory is collision-free for an agent whose volume is enlarged by ε and the robot stays within ε of it) with the assumption that all interacting agents maintain a constant velocity. Avoidance of dynamic obstacles was shown by [2], we reproduce it for the case of a dynamic obstacle in $\mathbb{R}^2$ (through it is extensible to $\mathbb{R}^3$). Let $\mathbf{p}(t)$ denote the position at time $t \geq t^k$. If not specified, variables are evaluated at t^k. Consider $||\mathbf{p}(t) - \mathbf{p}_j(t)|| =$

$$||f(\mathbf{z}, \mathbf{u}, \tilde{t}) - (\mathbf{p}_j + \mathbf{v}_j\tilde{t})|| \underset{\mathbf{u}\in\hat{R}(\mathbf{z},\varepsilon)}{\geq} ||(\mathbf{p} + \mathbf{u}\tilde{t}) - (\mathbf{p}_j + \mathbf{v}_j\tilde{t})|| - \varepsilon \underset{\mathbf{u}\in A_j(\mathbf{p},\varepsilon)}{\geq} r + r_j$$

For avoidance of static obstacles, we have that $\mathbf{u} \in F(\mathbf{p}, \varepsilon)$ implies, for all $\tilde{t} \in [0, \tau]$,

$$\mathbf{u} \in F(\mathbf{p}, \varepsilon) \underset{\text{Alg. 2, } P \text{ convex}}{\Rightarrow} (\mathbf{p} + \mathbf{u}\tilde{t}) \notin \mathcal{O} + V_\varepsilon \underset{\mathbf{u}\in\hat{R}(\mathbf{z},\varepsilon)}{\Rightarrow} f(\mathbf{z}, \mathbf{u}, \tilde{t}) \notin \mathcal{O} + V.$$

If (7) is *infeasible*, no collision-free solution exists that respects all of the constraints. Since the time horizon is larger than the required time to stop, passive safety is preserved by slowing down on the last feasible path and eventually reaching a stop. Also, since this computation is performed at a high frequency, each individual robot is able to adapt to changing situations, and the resulting motion is collision-free.

Realizes the reactive task specification. Since the local planner is myopic, it provides guarantees up to a time horizon τ and consequently may result in deadlock and livelock. However, as we have shown, the planner's local guarantees allow a discrete abstraction that the high-level strategy can use to resolve deadlocks and avoid livelocks. Here we formally prove the guarantees on the high-level behavior provided by our synergistic online and offline synthesis.

Proposition 1 *Given a task specification φ that ignores collisions, if the resulting specification φ' defined in Sect. 4 is realizable, then the corresponding strategy automaton also realizes φ.*

Proof Assume given $\varphi = \varphi_i^e \wedge \varphi_t^e \wedge \varphi_g^e \implies \varphi_i^s \wedge \varphi_t^s \wedge \varphi_g^s$. Recall that $\varphi' = \varphi_i^e \wedge \varphi_t^e \wedge \varphi_g^e \implies [\varphi_i^s]' \wedge [\varphi_t^s]' \wedge \varphi_g^s$, where $[\varphi_i^s]'$ and $[\varphi_t^s]'$ contain φ_i^s and φ_t^s as subformulas, respectively. Suppose that strategy automaton $\mathcal{A}_{\varphi'}$ realizes φ'. This means that the resulting controller is guaranteed to fulfill the requirement $[\varphi_i^s]' \wedge [\varphi_t^s]' \wedge \varphi_g^s$ as long as the environment fulfills the assumption $\varphi_i^e \wedge \varphi_t^e \wedge \varphi_g^e$. This implies that $A_{\varphi'}$ fulfills $\varphi_i^s \wedge \varphi_t^s \wedge \varphi_g^s$ as long as the environment fulfills the assumption $\varphi_i^e \wedge \varphi_t^e \wedge \varphi_g^e$. □

Proposition 2 *Given a task specification φ that ignores collisions, if φ is realizable but the resulting specification φ' is not realizable, then the revision procedure in Sect. 4.2 will find an assumption φ_{rev}^e to add to φ'.*

Proof Suppose φ is realizable by strategy $\mathcal{A}_\varphi$, but φ' is not realizable, admitting counterstrategy $\mathcal{C}_{\varphi'} = (\mathcal{Q}, \ldots)$. It suffices to show that the set S_{cuts} is nonempty. Assume for a contradiction that S_{cuts} is empty. Then the rising edge of deadlock θ_s^i

never occurs for any i, so no robot transitions are ever disabled. Since we assume that deadlock does not occur in the initial state, this means that x^i is always `False` for every i. Therefore $[\varphi_i^s]' \wedge [\varphi_t^s]' \wedge \varphi_g^s$ defined in Sect. 4 reduces to $\varphi_i^s \wedge \varphi_t^s \wedge \varphi_g^s$. The lack of deadlock means that any region transition contained in $\mathcal{A}_\varphi$ is still admissible, and therefore $\mathcal{A}_\varphi$ can be used as a strategy to realize φ'.

Note that it may be the case that S_{cut} is nonempty, but for every $(p, q) \in S_{cuts}$, the resulting revision $(\psi_{\mathcal{Y}}(p) \wedge \psi_{\mathcal{X}}(p) \implies \neg \bigcirc \psi_{\mathcal{X}}(q))$ contradicts φ_e^t. This indicates that φ is only realizable because it makes unreasonable assumptions on the environment. Our approach identifies this fact as a by-product of the revision process.

Computational complexity. The high-level reactive synthesis is exponential in the number of propositions [4], which scales linearly with n_{robots} – no worse than existing approaches (e.g. [15]). When one or more dynamic obstacles are considered, the number of propositions does not depend on the number of dynamic obstacles.

For the online component, a convex program is solved independently for each robot, with the number of constraints linear in the number of neighboring robots. The run-time of the iterative computation of the convex volume in free space barely changes with the number of obstacles, up to tens of thousands [7], and a timeout can be set, with the algorithm returning the best solution found.

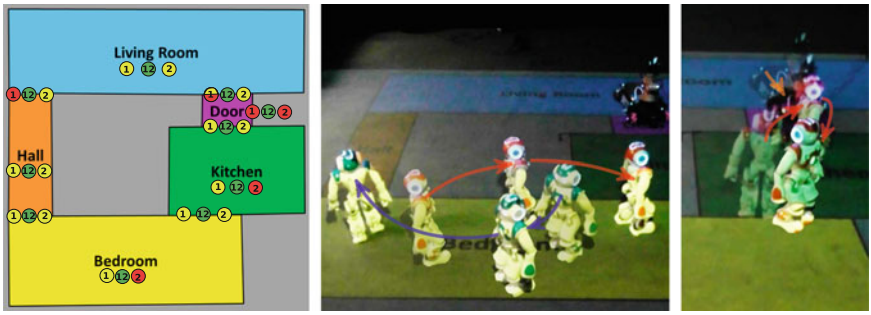
7 Experiments and Simulations

The synthesis procedure described in Sect. 4 was implemented with the `slugs` synthesis tool [8], and executed with the LTLMoP toolkit [9]. The local motion planner, Sect. 5, was implemented with the IRIS toolbox [7] and an off-the-shelf convex optimizer. We consider the dynamic obstacles to be cooperative in avoiding collisions. A video is available at http://youtu.be/esa3osYtvGA.

7.1 Humanoid Robots

We synthesize a controller for a "garbage collection" scenario, carried out by two humanoid robots (able to rotate in place, move forward and along a curve) occupying the workspace in Fig. 3a. The robots are required to patrol the *Living Room* (R_{LR}) and *Bedroom* (R_{BR}) [$\square\diamondsuit(\pi^1_{LR}) \wedge \square\diamondsuit(\pi^1_{BR}) \wedge \square\diamondsuit(\pi^2_{LR}) \wedge \square\diamondsuit(\pi^2_{BR})$] and if *garbage* is observed, pick it up [$\square(\pi^1_{garb} \implies \pi^1_{act,pickup}) \wedge \square(\pi^2_{garb} \implies \pi^2_{act,pickup})$]. The robots must always avoid other moving agents.

The system propositions are actions to move between regions ($\pi^i_{act,LR}$, $\ldots, \pi^i_{act,BR}$) and to pick up ($\pi^i_{act,pickup}$). The environment propositions are sensed garbage (π^i_{garb}), region completions ($\pi^i_{LR}, \ldots, \pi^i_{BR}$), and pick up completion (π^i_{pickup}). We omit the encoding of Definition 2, though these conditions are implied.



(a) Workspace with specification revisions (b) Avoidance maneuver (c) Deadlock resolution

Fig. 3 Planar scenario with two centrally-controlled Nao robots and a dynamic obstacle (youBot). **a** Workspace showing specification revisions for each region completion/activation pairs where singleton or pairwise deadlock may occur. Dot placement corresponds to the $AP^{act}_{\mathcal{R}}$ for each region; numbers indicate the robot(s) that are allowed to be in deadlock, and color represents the necessary restrictions on deadlock. *Green* dots represent transitions where deadlock is allowed; *yellow dots* where deadlock is allowed, but only up to a finite time; and *red dots* where deadlock is not allowed. **b** and **c** Three consecutive frames of the video are superimposed. In (**c**), one of the Naos reverses direction to resolve the deadlock with the youBot

Our synthesis tool took 84 seconds yielding an automaton with 5836 states and four memory propositions.

A graphical representation of the revisions produced by our algorithm is shown in Fig. 3a. The red dots indicate that dynamic obstacles should not produce deadlock when the robot is making the indicated transition. We also alert the user via textual feedback – one of the generated statements for our scenario is: `Deadlock should not occur when robot 1 is in the Hall moving toward the Living Room.` We employ two Aldebaran Nao robots and a teleoperated KUKA youBot as the dynamic obstacle. As demonstrated in the snapshots in Fig. 3, the Naos are capable of executing the task, avoiding collision and resolving deadlocks.

We further evaluated the approach in simulation, from 10 different initial conditions and with two dynamic obstacles. No collisions occurred and the approach was able to resolve 47 deadlock events out of the 51 encountered. Those that could not be resolved occurred in disallowed transitions labeled red in Fig. 3a.

7.2 Scalability with Respect to Dynamic Obstacles

Considering the example in Sect. 7.1, the synthesized controller for two robots consists of 29 propositions, and is invariant to the number of dynamic obstacles. In comparison, we consider a baseline approach similar to Fig. 1a without a local planner where one-cell separation with other robots and dynamic obstacles (DO) is kept. In this case, 20 propositions are required for zero DO, 25 for one DO, 30 for two DO,

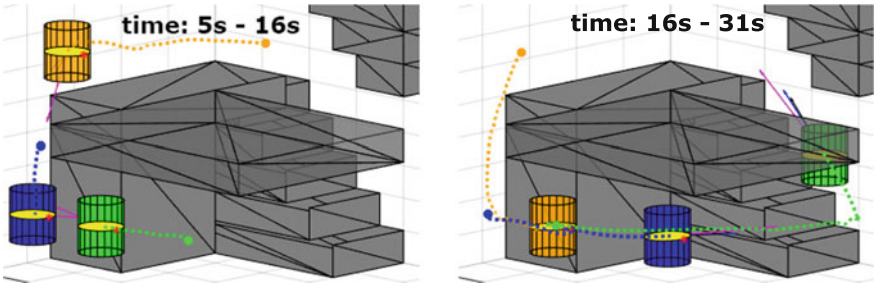


Fig. 4 Deadlock resolution (*green* robot) and safe navigation in a 3D environment. Quadrotors are displayed at the final time and their paths for the time interval. Each *yellow* disk represents a quadrotor and the cylinder its safety volume. The *orange* robot represents the dynamic obstacle

35 for three DO and 80 propositions for eight DO. Because the obstacles are assumed to behave adversarially, they can violate mutual exclusion if they enter to within a neighboring region to the robot. Hence, the synthesis procedure is *not realizable* for one or more dynamic obstacles. Our approach, on the other hand, is realizable independently of the number of dynamic obstacles and requires fewer propositions than the case with two or more DO.

7.3 Quadrotors

We next demonstrate the effectiveness of the approach in a 3D scenario with the $5 \times 5 \times 5\,\mathrm{m}^3$ two floor workspace shown in Fig. 4, where robots can move between floors through a vertical opening at the left corner or the stairs at the right side of the room. We simulate, using the model described in [2], two controlled quadrotors, and one more as a dynamic obstacle. The task is to infinitely often visit the top and bottom floors while avoiding collisions and resolving deadlock. The high-level controller is synthesized as described in Sect. 4. A local planner for the 3D environment is constructed following Sect. 5. A representative experiment is shown in the snapshots in Fig. 4. The green robot enters deadlock when moving towards the upwards corridor; however, deadlock is resolved by taking the alternative route up the stairs.

8 Conclusion

We present a framework for synthesizing a high-level finite state machine and collision-free local planner that guarantees completion of a task specified in linear temporal logic, where we consider high-level specifications that are able to capture basic locomotion, sensing and actuation capabilities. Our approach is less conserva-

tive than current approaches that impose a separation between agents, and is computationally cheaper than explicitly modeling all possible obstacles in the environment. If no controller is found that satisfies the specification, the approach automatically generates the needed assumptions on deadlock to render the specification realizable and communicates these to the user. The approach generates controllers that accommodate deadlock between robots or with dynamic obstacles *independently of* the precise number of obstacles present, and we have shown that the generated controllers are correct with respect to the original specification. Experiments with ground and aerial robots demonstrate collision avoidance with other agents and obstacles, satisfaction of a task, deadlock resolution and livelock-free motion. Future work includes optimizing the set of revisions found, and decentralizing the synthesized controller.

Acknowledgements This work was supported in part by NSF Expeditions in Computer Augmented Program Engineering (ExCAPE), ONR MURI Antidote N00014-09-1-1031, SMARTS N00014-09-1051, the Boeing Company and TerraSwarm, one of six centers of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA.

References

1. Alonso-Mora, J., Gohl, P., Watson, S., Siegwart, R., Beardsley, P.: Shared control of autonomous vehicles based on velocity space optimization. In: 2014 IEEE International Conference on Robotics and Automation (ICRA), pp. 1639–1645 (2014)
2. Alonso-Mora, J., Naegeli, T., Siegwart, R., Beardsley, P.: Collision avoidance for multiple aerial vehicles. Auton. Robot. (2015)
3. Alur, R., Moarref, S., Topcu, U.: Counter-strategy guided refinement of gr(1) temporal logic specifications. In: Formal Methods in Computer-Aided Design (FMCAD), pp. 26–33 (2013)
4. Bloem, R., Jobstmann, B., Piterman, N., Pnueli, A., Sa'ar, Y.: Synthesis of reactive (1) designs. J. Comput. Syst. Sci. **78**(3), 911–938 (2012)
5. Chen, Y., Ding, X.C., Stefanescu, A., Belta, C.: Formal approach to the deployment of distributed robotic teams. IEEE Trans. Robot. **28**(1), 158–171 (2012)
6. DeCastro, J.A., Ehlers, R., Rungger, M., Balkan, A., Tabuada, P., Kress-Gazit, H.: Dynamics-based reactive synthesis and automated revisions for high-level robot control. In: CoRR (2014)
7. Deits, R., Tedrake, R.: Computing large convex regions of obstacle-free space through semi-definite programming. In: Workshop on the Algorithmic Fundamentals of Robotics (2014)
8. Ehlers, R., Finucane, C., Raman, V.: Slugs gr(1) Synthesizer (2013). http://github.com/ltlmop/slugs
9. Finucane, C., Jing, G., Kress-Gazit, H.: Ltlmop: Experimenting with language, temporal logic and robot control. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (2010)
10. Kress-Gazit, H., Fainekos, G.E., Pappas, G.J.: Temporal logic based reactive mission and motion planning. IEEE Trans. Robot. **25**(6), 1370–1381 (2009)
11. Li, W., Dworkin, L., Seshia, S.A.: Mining assumptions for synthesis. In: 9th IEEE/ACM International Conference on Formal Methods and Models for Codesign, MEMOCODE (2011)
12. Livingston, S.C., Prabhakar, P., Jose, A.B., Murray, R.M.: Patching task-level robot controllers based on a local μ-calculus formula. In: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA). Karlsruhe, Germany (2013)
13. Raman, V., Kress-Gazit, H.: Explaining impossible high-level robot behaviors. IEEE Trans. Robot. **29**(1), 94–104 (2013). doi:10.1109/TRO.2012.2214558

14. Raman, V., Piterman, N., Kress-Gazit, H.: Provably correct continuous control for high-level robot behaviors with actions of arbitrary execution durations. In: IEEE International Conference on Robotics and Automation (2013)
15. Ulusoy, A., Smith, S.L., Ding, X.C., Belta, C., Rus, D.: Optimality and robustness in multi-robot path planning with temporal logic constraints. I. J. Robot. Res. **32**(8), 889–911 (2013)
16. Vardi, M.Y.: An automata-theoretic approach to linear temporal logic. In: Logics for concurrency, pp. 238–266. Springer, Heidelberg (1996)
17. Wong, K.W., Ehlers, R., Kress-Gazit, H.: Correct high-level robot behavior in environments with unexpected events. In: Proceedings of Robotics: Science and Systems (2014)
18. Wongpiromsarn, T., Ulusoy, A., Belta, C., Frazzoli, E., Rus, D.: Incremental synthesis of control policies for heterogeneous multi-agent systems with linear temporal logic specifications. In: IEEE International Conference on Robotics and Automation (ICRA) (2013)